APPENDIX

```
while(1) {

Step 1:
      DEM::findProaction();

Step 2:
      selection = DEM::waitForInterrupt(&rfd);

Step 9:
      DEM::updateTimeVariables (selection);

Step 3:
      if (firingDelay > 0)
            for(i = 0; i < realtimeComponents.size; i++)
                  realtimeComponents.elements[i]->update(firingDelay);

Step 5:
      if (interruptAlerts.size > 0)
            alert = DEM::getInterruptAlert();
      else if (selection < 0) {
            // selection < 0: some error with select happened.
            writeErrorLog(error);
            continue;
      }

      else if ((selection == 0) && (interruptAlerts.size == 0))
            // if selection is 0 and no interrupt was received (timeout has
            // expired), a fake alert is generated.
            alert = DEM::getAlert();

Step 4:
      else if (shellIn >= 0 && FD_ISSET(shellIn, &rfd)) {
            // selection > 0 + shellIn: a shell command has been received.
            DEM::executeShellCommand();
            continue;
      }

      else if (incomingConnectionRequestPort >= 0
            && FD_ISSET(incomingConnectionRequestPort, &rfd)) {
            // selection > 0 + incomingConnectionRequestPort: a new
            // connection is being requested.
            struct sockaddr_in client;
            int l = sizeof(client);

            SOCKET c = accept(incomingConnectionRequestPort,
                  (struct sockaddr *)&client, &l);
            if (c < 0) {
                  writeErrorLog(error);
                  continue;
            }
            else {
                  // The address of the client is stored in
                  // DEM::pendingConnections
                  pendingConnections.add(c);
                  continue;
            }
      }
      else {
```

```
Step 6:
            // selection > 0 + message on active connection or pending
            // connection

            // Every activeConnection is checked.
            for (i=0; i<activeConnections.size; i++)
                  if (FD_ISSET(activeConnections.elements[i]->fd, &rfd))
                        break;
                  if (i == activeConnections.size) {
                        // If no activeConnection was selected, every
                        // pendingConnection is checked.
                        int j;
                        for(j = 0; j < pendingConnections.size; j++)
                              if (FD_ISSET(pendingConnections.elements[j],
&rfd))
                                    break;
                              if (j == pendingConnections.size) {
                                    writeErrorLog("DEM::received interrupt
but no inputs are set\n");
                                    continue;
                              }
                              else
                                    // Detect the protocol and validate the
                                    // message

      DEM::validatePendingConnection(SocketType,
                                    pendingConnections.elements[j]);
                              continue;
                  }
                  else
                        // Get the alert from the selected activeConnection.
                        alert = DEM::getAlert(SocketType,
activeConnections.elements[i]->fd);
      }

      if (alert == 0)
            continue;

Step 7:
      // The action associated with the scheduled transition is executed and
      // the return event is stored in event.
      Event *event = (scheduledComponent->*scheduledComponent->
            a()[scheduledTransition])(alert);

Step 8:
      if ((scheduledComponent->q
            = (scheduledComponent->t()[scheduledTransition]).to) < 0)
            // If the component has reached the stop state, add it to
            // DEM::deletedComponents
            deletedComponents.add(scheduledComponent);
      else {
            // Otherwise execute the flow function. The scheduledComponent
            // is added to DEM::changedComponents.
            (scheduledComponent->*scheduledComponent->
                  f()[scheduledComponent->q])();
            if (scheduledComponent->transientStates()[scheduledComponent->q])
                  transientStateComponents.add(scheduledComponent);
            else
                  transientStateComponents.remove(scheduledComponent);
            changedComponents.add(scheduledComponent);
      }
      // No return event.
      if (event == 0) {
            transitionId += 1;
            DEM::writeTransitionLog(alert, event);
            delete alert;
            for(i=0; i<deletedComponents.size; i++)
                  DEM::stopComponent(deletedComponents.elements[i]);
            continue;
      }

      // Propagate the return event to dependents.

      // Find the dependents.
      for(i=0;
      i < scheduledComponent->dependents[scheduledTransition]->size;
      i++)
            propagatedComponents.add
            (scheduledComponent->dependents[scheduledTransition]-
>elements[i]);

      // Find enabled responses in dependents, execute them and do
      // housekeeping.
      for(i=0; i<propagatedComponents.size; i++) {
            Component *c = propagatedComponents.elements[i];
            int t = c->findResponse(event->event);
            if (t<0)
                  continue;

            Event *re = (c->*c->a()[t])(event);
            if (re != 0 && re != event)
                  responseEvents.add(re);

            if ((c->q = c->t()[t].to) < 0)
                  deletedComponents.add(c);
            else {
                  (c->*c->f()[c->q])();
                  if (c->transientStates()[c->q])
                        transientStateComponents.add(c);
                  else
                        transientStateComponents.remove(c);
                  changedComponents.add(c);
            }
      }

Step 10:
      transitionId += 1;
      DEM::writeTransitionLog(alert, event);

      // Final housekeeping.
      delete alert;
      if (event != alert) // event could be just "return alert"
            delete event;
      for(i=0; i<responseEvents.size; i++)
            delete responseEvents.elements[i];
      for(i=0; i<deletedComponents.size; i++)
            DEM::stopComponent(deletedComponents.elements[i]);
  }
```